PRÁCTICA DE
LENGUAJES DE PROGRAMACIÓN
JUNIO 2011

Alumno: Manuel Jesús Canga Muñoz

Email: manuelcanga@gmail.com

1.DESCRIPCIÓN

Monopoly es un juego de mesa que como el nombre sugiere, el objetivo del juego es hacer un monopolio de oferta, poseyendo todas las propiedades inmuebles que aparecen en el juego. Los jugadores mueven sus respectivas fichas por turnos en sentido horario alrededor de un tablero, basándose en la puntuación de los dados, y caen en propiedades que pueden comprar de un banco imaginario. Si las propiedades en las que caen ya tienen dueños, los dueños pueden cobrar alquileres o quien caiga podrá comprárselas.

2.DETALLES GENERALES DE IMPLEMENTACIÓN

El código se ha estructurado en forma de paquetes para que cada paquete sirva de encapsulación con respeto al resto del programa así como para tener una cierta organización. A continuación listaremos cada paquete y en para cada uno los elementos que intervienen. Asi como también mostraremos un diagrama(también disponible también como fichero aparte para mejorar su vsualización) para ver su relación entre todos los elementos del paquete

3.Estructura:

La aplicación se distribuye en un zip, de tal forma:

/Monopoly.jar	→ archivo jar del juego
/docs/	→ documentación del juego
/docs/Monopoly.pdf	→ Esta documentación
/docs/jpgs	→ Imagenes de diagramas utilizados en este documento.
/source	→ Código fuente del juego

Monopoly.jar, es un archivo comprimido de tipo .jar que contiene todos los demás elementos dicho.

3.Modo de ejecución:

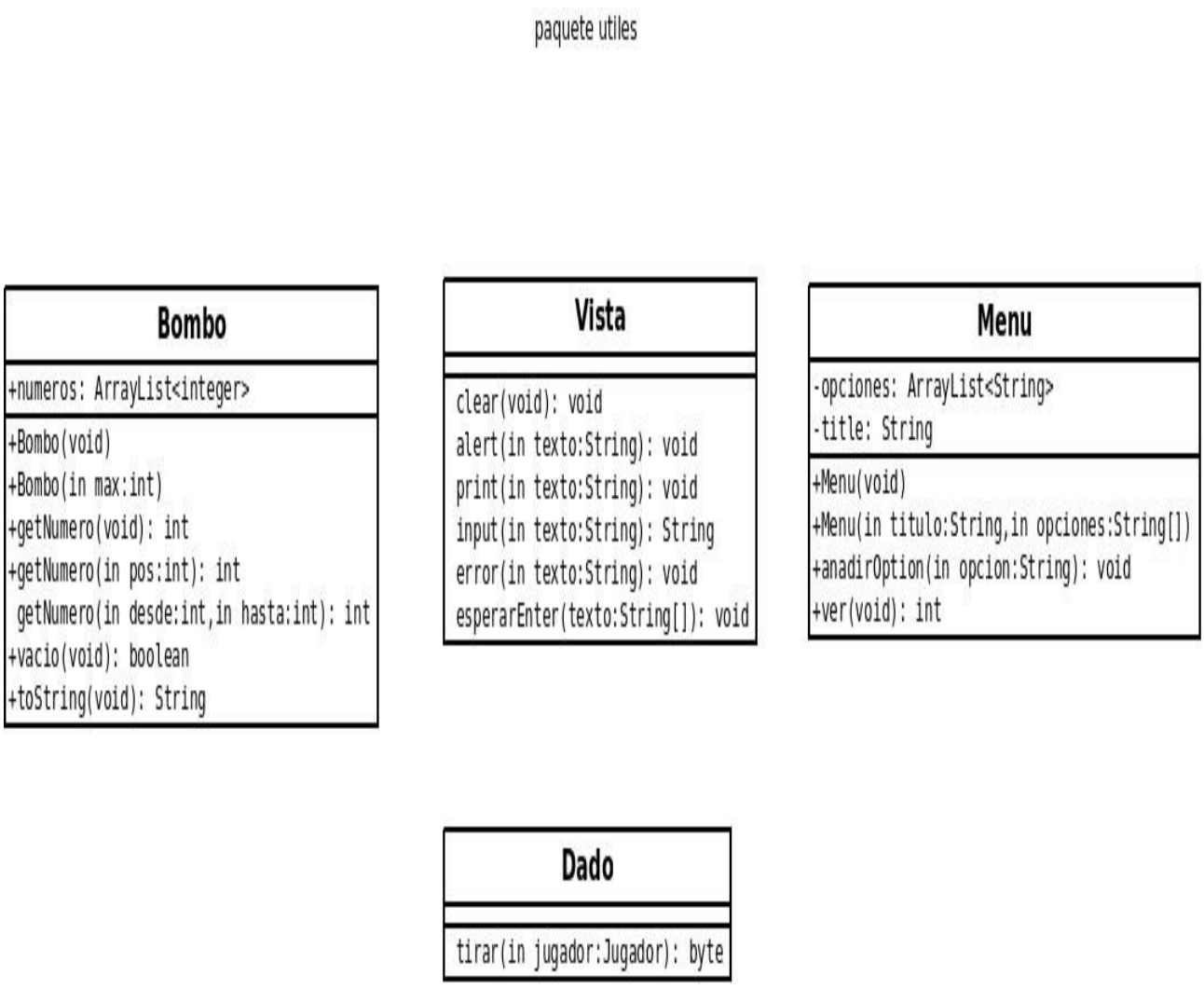
Desde línea de comando linux o windows se puede lanzar el juego de la siguiente manera(siempre que se tenga instalada la última versión jdsk de java):

java -jar Monopoly.jar

4.Desarrollo incremental:

El programa se ha realizado siguiendo el llamado 'desarrollo incremental', así que se irá mostrando los detalles del programa siguiendo los distintos pasos que se realizaron en su creación. Estos pasos concuerdan con los paquetes que forman parte del programa. Empezando por el primero de ellos y el más independiente “utiles” hasta el último que son los distintos tipos de casillas.

5. LISTADO DE PAQUETES DEL MONOPOLY



Estas clases sirven como herramientas para la creación del sistema de juegos.

Dado: Es una clase que representa un dado de juego, es estático porque sólo hay uno en toda la partida. Recibe el jugador que realizará la tirada.

Config: Establece un sistema de configuración del juego, permitiendo modificar algunos parámetros de este mediante el archivo Monopoly.conf

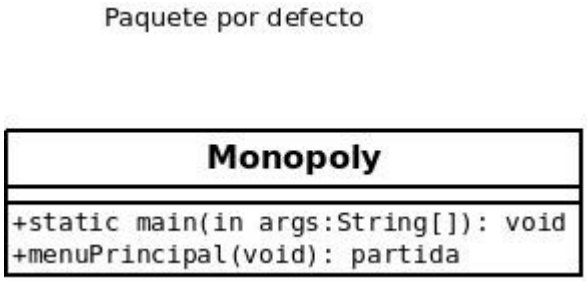
Vista: Constituye la interfaz de interacción entre jugadores y programa. Sus funciones son limpiar la pantalla, mostrar mensajes, pedir valores, … es un procedimiento estático para uso de manera rápida y fácil desde cualquier parte del juego, además porque no es necesario que guarde estados intermedios entre ejecución de un método u otro. Nos permite una abstracción de GUI, por si en el futuro, por ejemplo, quisiéramos hacer el juego a través de ventanas

Menú: Utilizando los elementos de la vista visto antes, represnta un menú en pantalla y espera la selección de una de las opciones. Así nos facilita la creación de menú.

Bombo: Asemeja a un bombo de lotería de navidad en el que se va sacando números

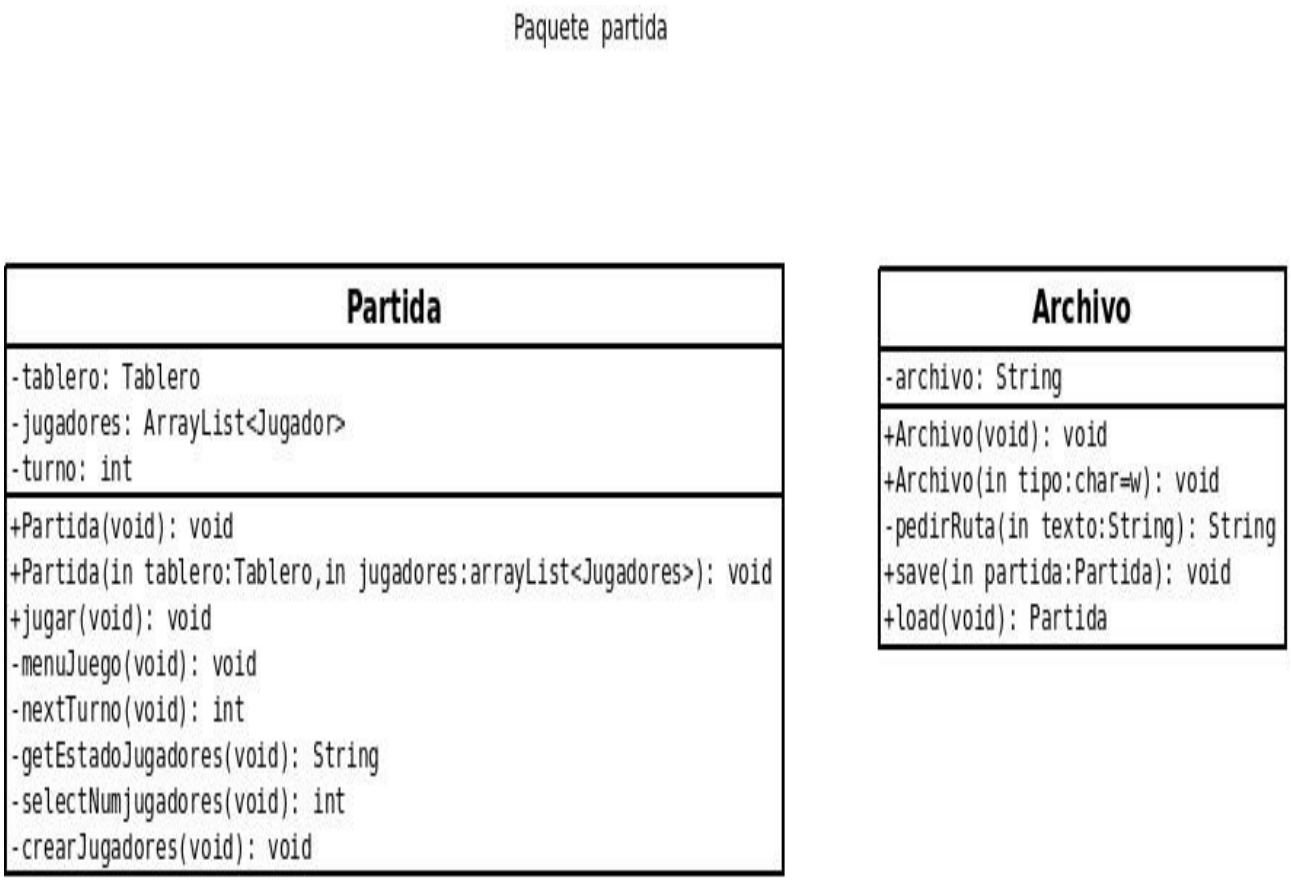
de aquellos que tiene almacenado. Sirve para obtener número al azar y sobretodo para obtener números de casillas no repetidos (muy útil para situar tipos de casillas en tablero).

- **sin paquete (o paquete por defecto)**



Monopoly: Esta clase se tiene un método estático *main*, que es el comienzo de la ejecución del programa Monopoly. También es el punto de partida para el juego de cara al usuario pues le muestra el menú principal de éste(función *menuPrincipal*).

- **partida**



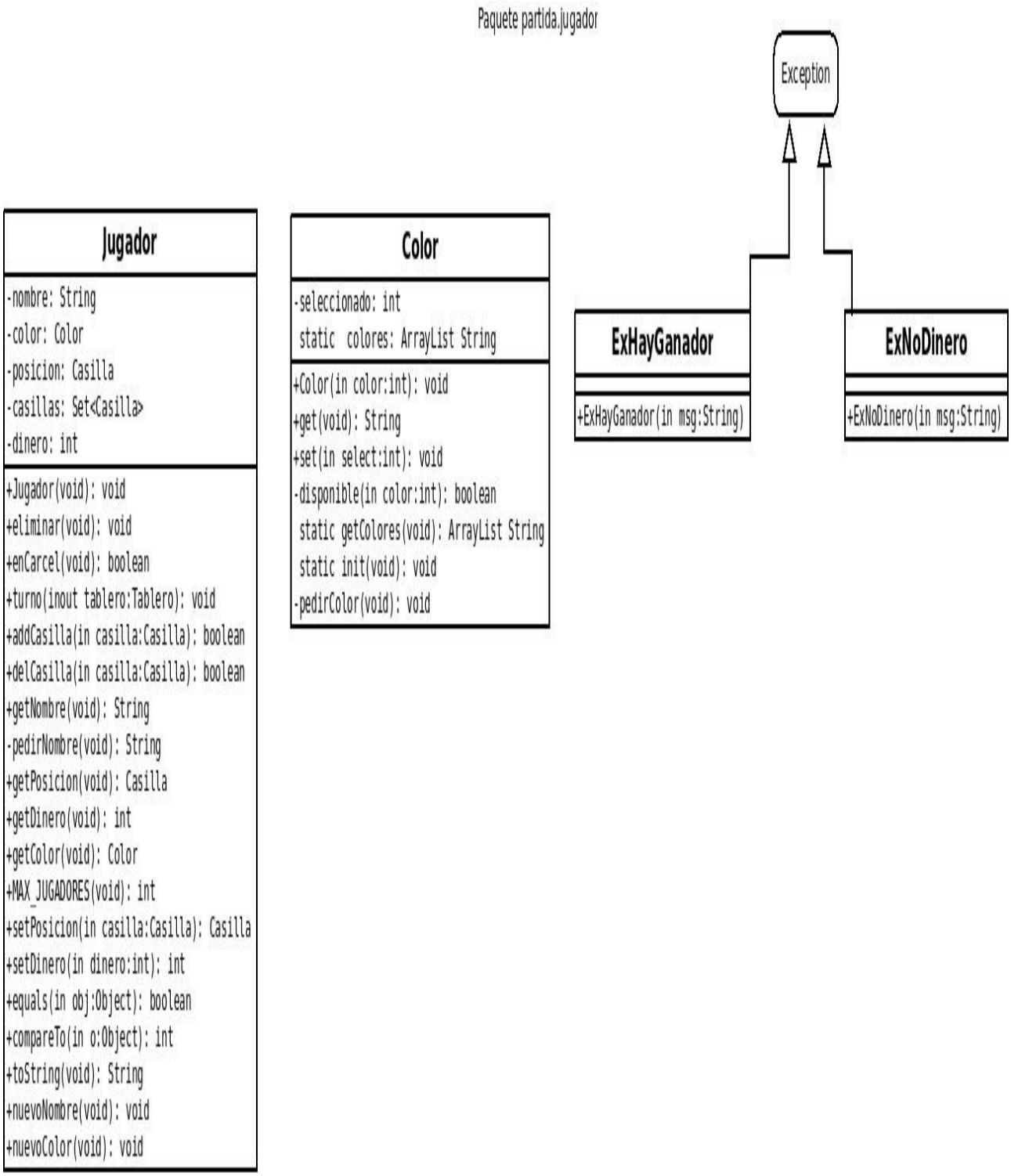
En este paquete encontramos lo referente a manejo de partidas de Monopoly.

Partida: Define una nueva partida y, por tanto, lleva todo lo asociado al desarrollo de esta: turno, jugadores que participan, tablero, el estado, menú de partida, etc

Archivo: Clase cuya utilización es para guardar partidas que se estén desarrollando(o sea, objetos partidas) o recupera partidas previamente guardadas(o sea, obtiene objetos partida). Para ello, se usa la serialización de la clase partida y de todos los elementos que contiene.

- **Partida.jugador**

Este paquete contiene todo los tipos de datos abstracto y excepciones asociados a los jugadores.



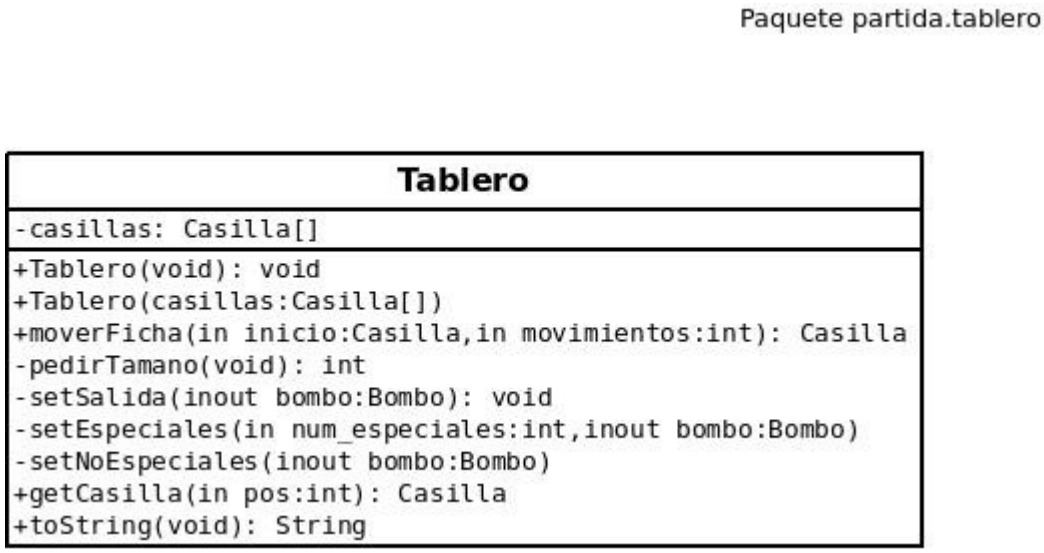
Jugador: Con todo lo necesario para gestionar jugadores.

Color: Es un tipo abstracto de datos color. Usa un atributo estático colores, para que cuando se instancie un nuevo color se valide si es uno de los colores disponible.Es decir, es un elemento único para todas las clases y mantiene los colores que es posible instanciar.

Excepción ExHayGanador: Se lanza cuando hay un nuevo ganador en el juego, es decir, detiene todo lo que se esté haciendo para proclamar a un jugador como ganador.

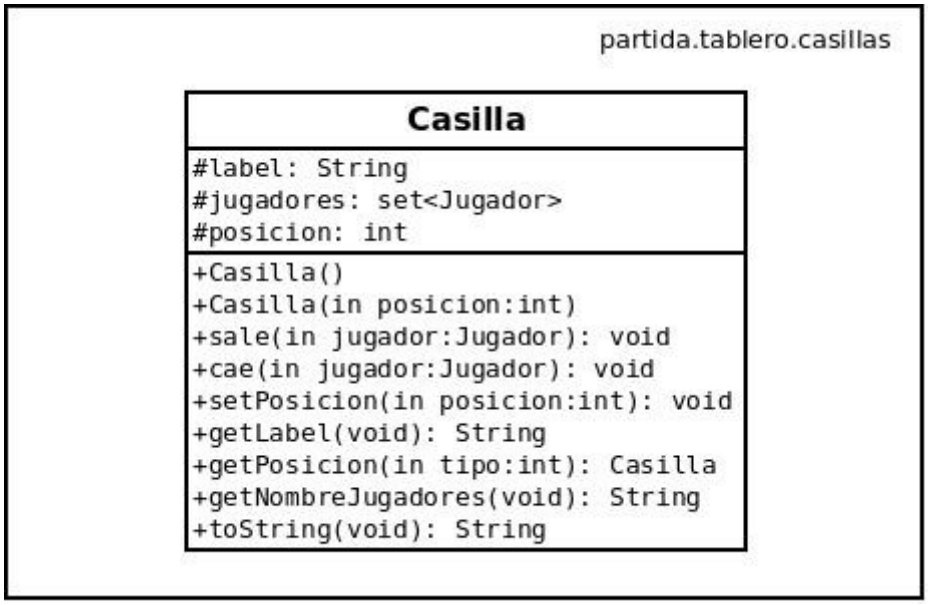
Excepción ExNoDinero: Se lanza cuando un jugador se queda sin dinero.

- ***partida.tablero***



Tablero: Gestiona las distintas casillas(casillas) que forman el tablero. Casillas es un array pues un tablero es de tamaño siempre fijo y es de tipo Casilla para utilizar las ventajas del polimorfismo.

partida.tablero.casillas



Casilla: Es un tipo de datos abstracto(TDA) que sirve de base para la creación del tablero. Además, es un patrón para los otros tipos de casillas. Nunca habrá casillas de tipo Casilla sólo sirve de definición por ello es de tipo abstracto.

El método *sale*, se refiere a cuando un jugador en su movimiento sale de una casilla(y por tanto, hay que quitarlo del conjunto de *jugadores* que se encuentra en dicha casilla, así como para otras tareas).

El método cae, se refiere a cuando un jugador después de tirar su dado termina cayendo una casilla del tablero.

Toda casilla tiene un *label*(su título), y *posicion*(lugar que ocupa dentro del tablero)

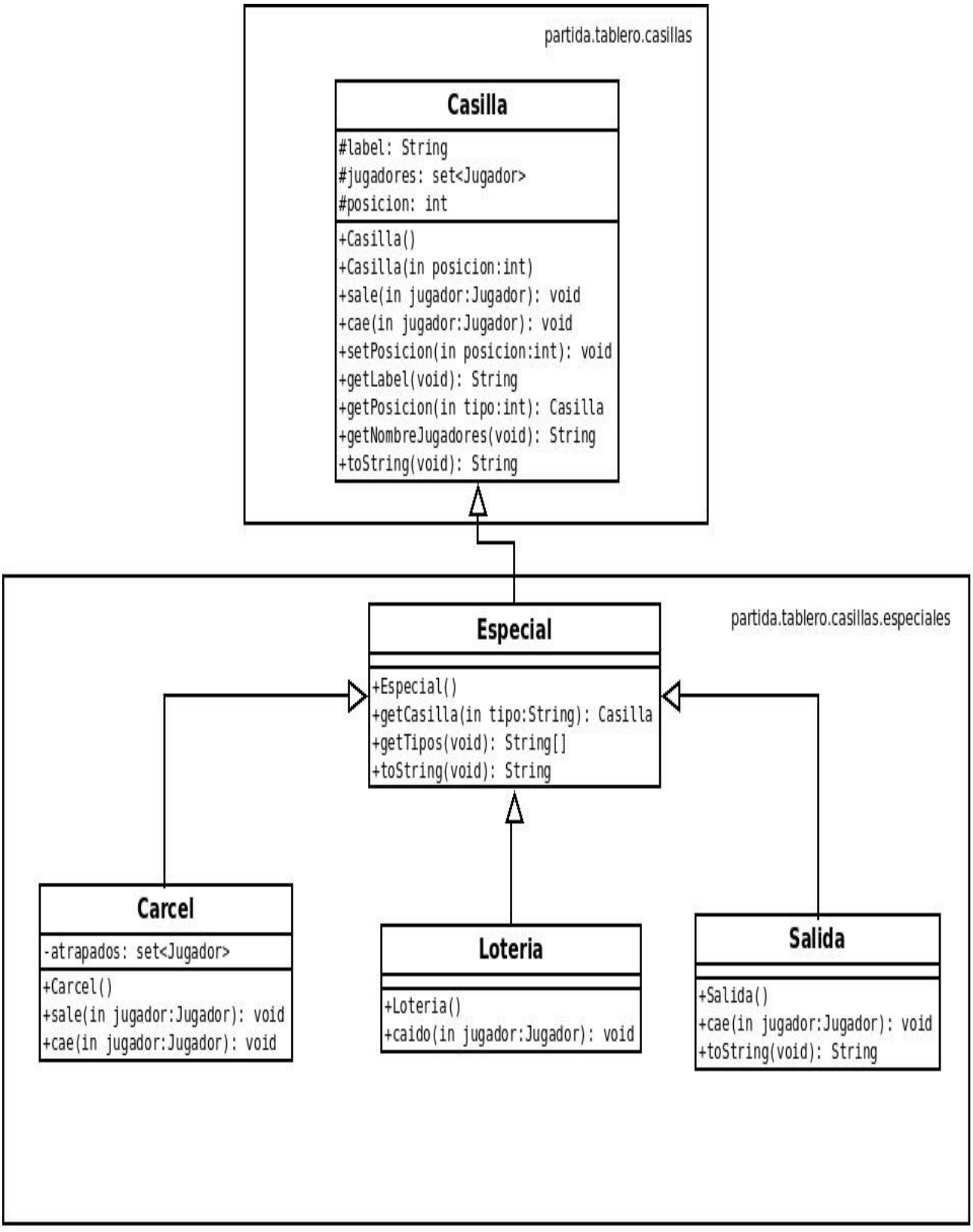
partida.tablero.casillas.especiales y partida.tablero.casillas.noespeciales

Especial y NoEspecial: *Como ocurría con Casilla, estas clases define unos patrones y unas bases para sus subtipos. Además, estos TDA sirve como una Factoría de sus subtipos. Una Factoría es un patrón de diseño por la cual una clase sirve para crear objetos de otros tipos distintos. En el caso de Especial y NoEspecial estas factorias nos facilita la creación de sus subtipos y encapsula el número de estos y el lugar donde se encuentren.*

En el paquete de especiales se han añadido los distintos tipos de casillas

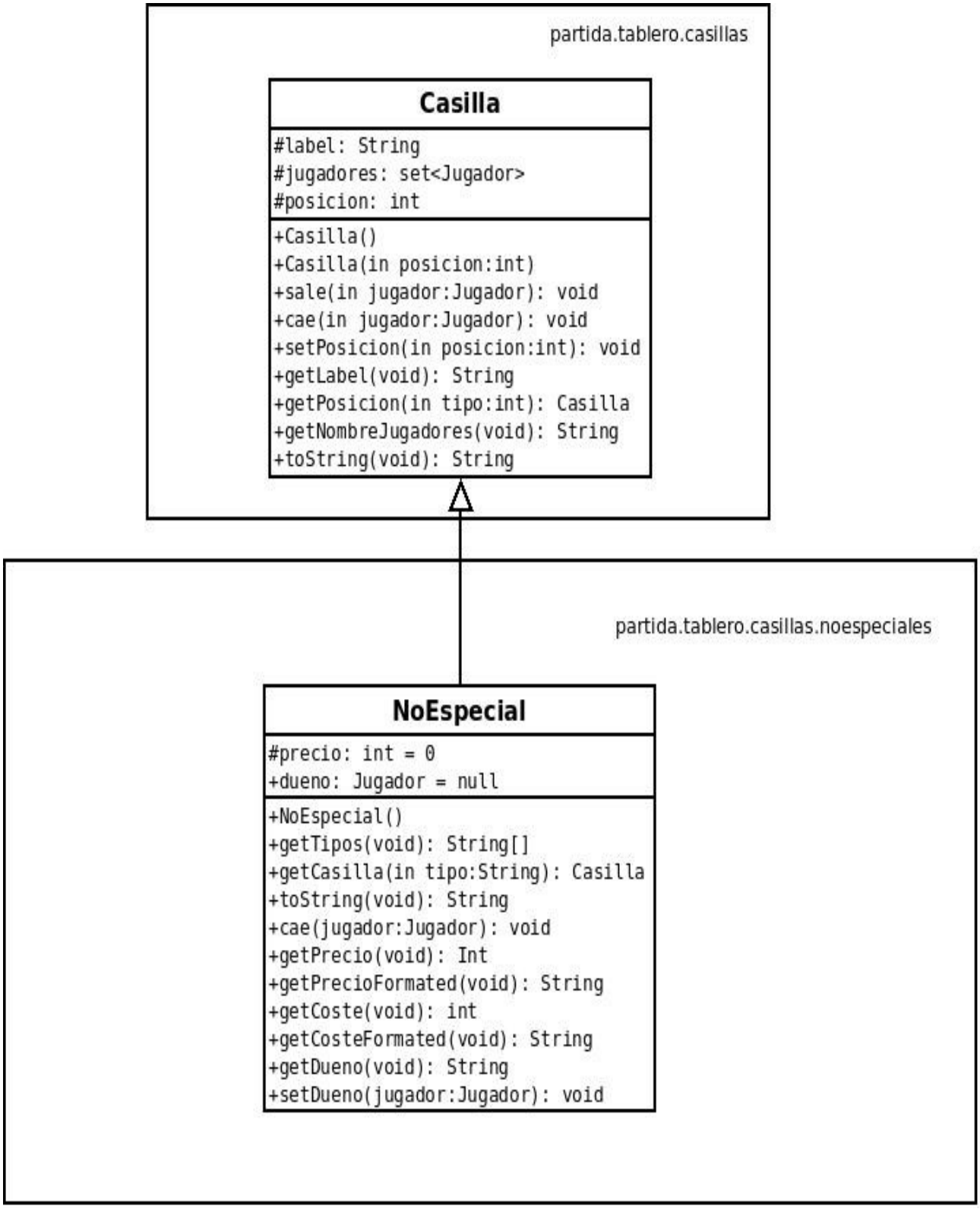
de este tipo: ***Carcel****,* ***Loteria*** *y* ***Salida****.*

partida.tablero.casillas.especiales

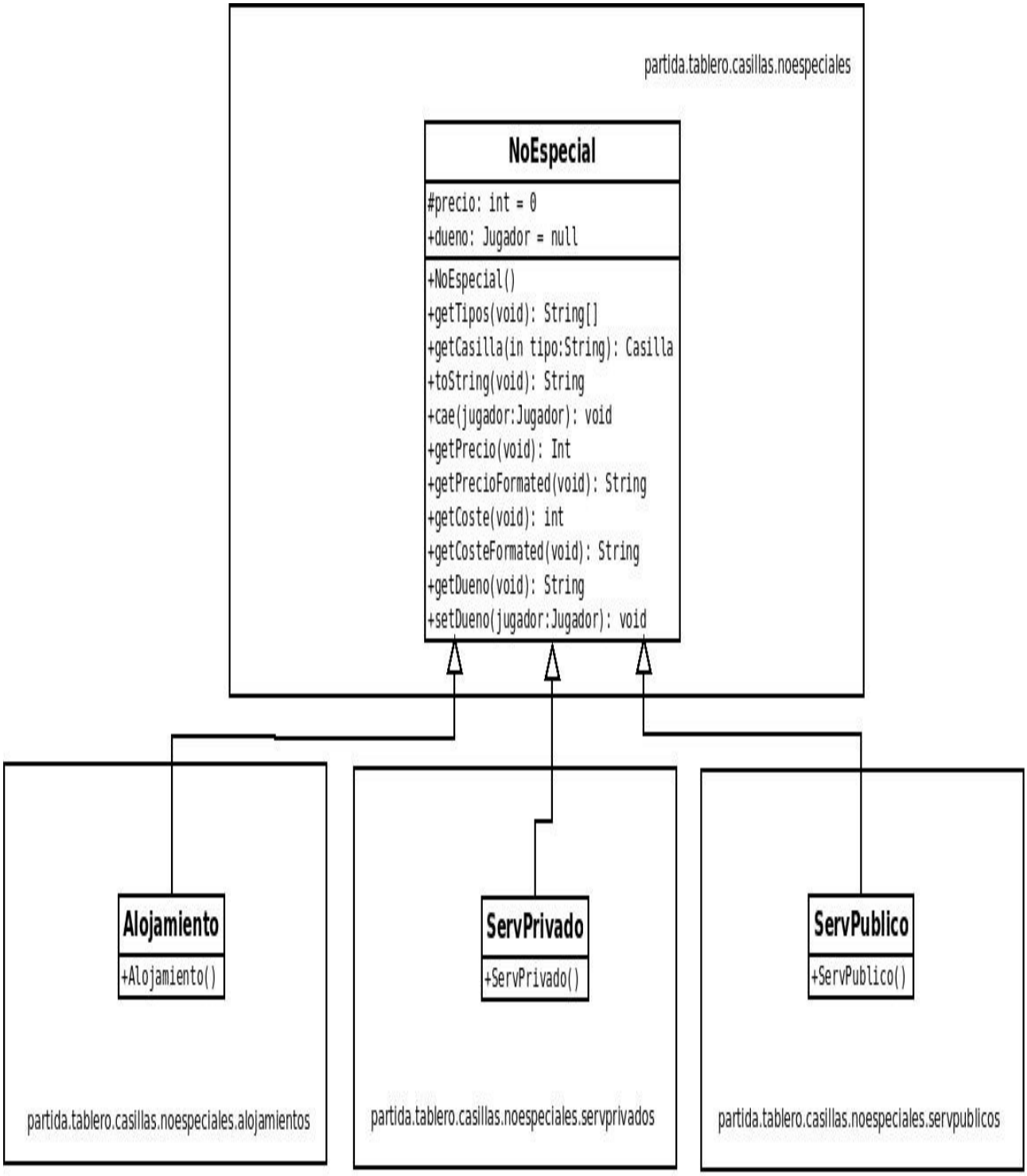


Partida.tablero.casillas.noespeciales

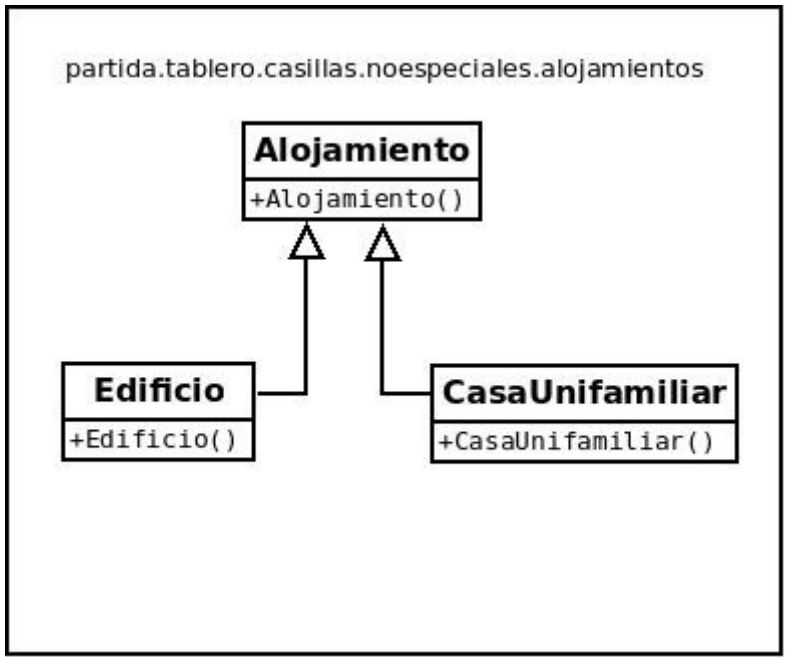
Los tipos de casillas que heredan de NoEspecial tienen la carácteristica de poseer un dueño y de tener un precio y un coste. El precio y coste se establece al azar en la inicialización de los objetos, estos variará según los subtipos.



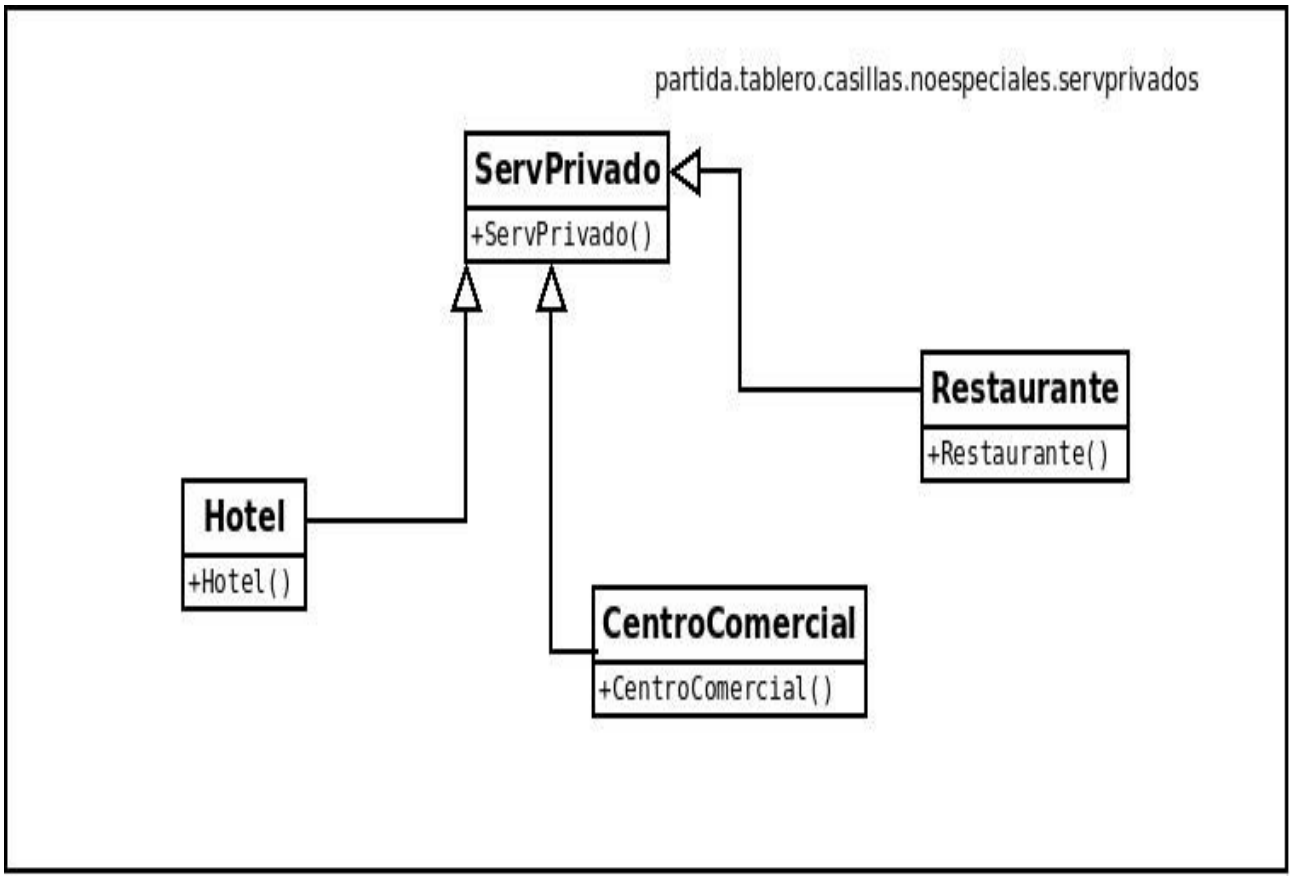
Las casillas NoEspeciales pueden ser de tres subtipos diferentes: alojamientos(*Alojamiento*), servicios privados(**ServPrivado**) y servicios públicos(*ServPublico*). Cada uno de estos subtipos así como los subtipos de estas están en paquetes diferentes por encapsulación y para mantener la jerarquía de subtipos.



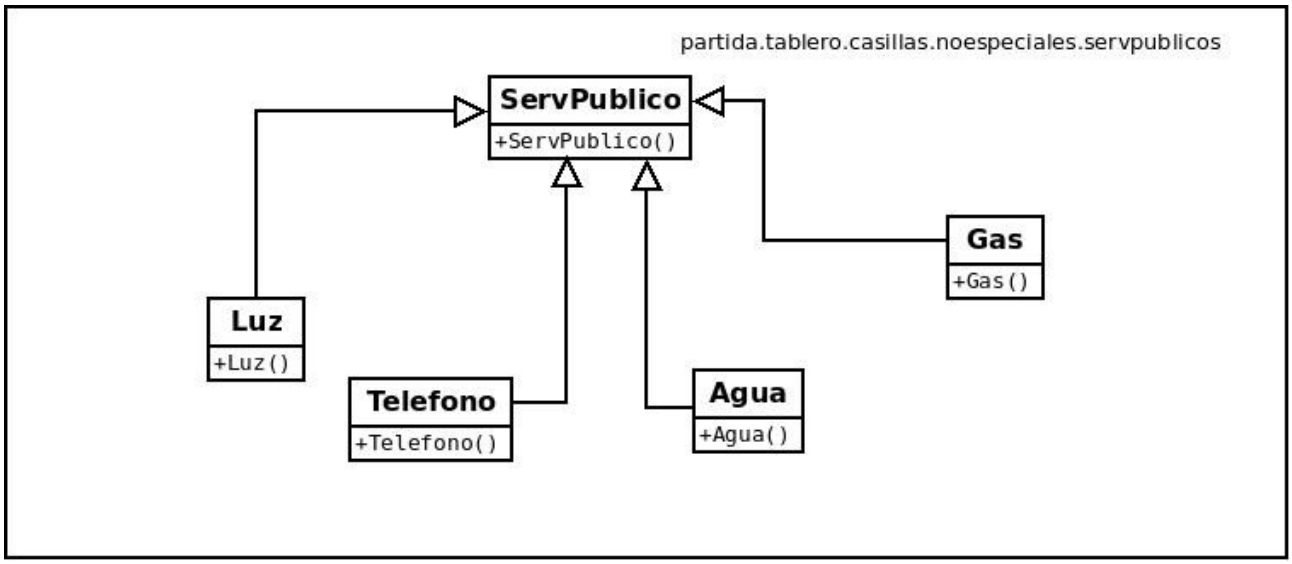
Partida.tablero.casillas.noespeciales.alojamientos



Partida.tablero.casillas.noespeciales.servprivados



Partida.tablero.casillas.noespeciales.servpublicos



Nota final: Me hubiera gustado incluir en esta parte el código fuente de toda la aplicación pero me ha sido imposible importarlo desde el eclipse a OpenOffice. Todo el código fuente se encuentra disponible en /src